PATENT #

[0001] White Space Algorithm to Support the Creation of high quality Computer-based Drawings Free of Graphics and Text overwrites.

Joint Inventors:

Name:	Dr. Anthony Bruce Crawford
Citizenship:	British
Residence:	230 West Lake Circle
	Madison, AL 35758
	USA

Name:	Dr. Edward Francis Boyle
Citizenship:	Irish
Residence:	5059 Sterling Grove Lane
	San Diego, CA 92130
	USA

BACKGROUND OF THE INVENTION

[0002] The problem of finding white space for the placement of graphics and text has long been acknowledged in the Computer Aided Design (CAD) industry. This is clearly evident when creating dense cartographic maps consisting of lines, curves, symbols and text. The White Space Algorithm provides capabilities for use by end-user applications to determine the quality of space available for adding graphics and text into a drawing view or onto a drawing plane.

[0003] This White Space Algorithm is applicable to any 3D and 2D computer based application that creates 2D drawing representations. The technology augments, and adds significant value to, solutions that cover generic CAD graphics and a broad range of applications including, but not limited to, structural engineering, civil engineering, transportation engineering, electrical engineering, plant design, architecture, facility management, mapping, utility management, emergency dispatch management and air traffic control displays.

[0004] This algorithm can also solve the difficult problem associated with scale changes on computer generated cartographic maps where annotation text must be adjusted manually after each change of scale. A further area of application is annotation of 2-D drawings that are extracted from 3-D CAD models where text and dimensioning typically needs to be adjusted manually to eliminate overwrites. Moreover, as drawings are often extracted multiple times as a result of model revisions, this manual process must be repeated to maintain consistency and drawing quality. Essentially, any diagramming application is able to benefit from this invention.

[0005] As part of a comprehensive structural engineering software application, the inventors have successfully applied the White Space Tracking Algorithm to automatically generate high quality computer-based drawings for steel fabrication layouts and details.

BRIEF SUMMARY OF THE INVENTION

[0006] The White Space Algorithm is an invention that provides the means for client applications to create drawings that are free from overwritten graphics and text elements. The solution is based on the creation, tracking and comparison of a series of bitmap images that represent 'committed ' and 'proposed' drawing plane graphics. These bitmaps are derived from a transformation of the vector graphics and text into the drawing plane and subsequent mapping into an equivalent pixel representation.

[0007] To begin White Space tracking and subsequent location of available space, a client application sets the value of control variables to guide the algorithm. Efficient bit level manipulations are then employed to compare bitmap pairs and return the quality of space via status variables. Adjustment schemes based on a concept of slide vectors are used to reposition proposed graphics into locations of white space.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The following drawings are part of the specification and these show features and concepts used by the White Space Algorithm:

Fig. 1 shows the pixel representation of a line after vector to pixel transformation using the standard Bresenham line drawing algorithm.

Fig. 2 shows the range of a text element and the definition of halos, slide vectors and text angle.

Fig. 3 shows the line skip over feature used to avoid dense areas of graphics.

Fig. 4 shows a pixel representation of a filled polygon after transforming the perimeter vectors with the Bresenham algorithm and applying a solid fill to the interior.

Fig. 5 shows the notion of cardinal points for text placement.

Fig. 6 shows how a placement point is adjusted through a rotation angle

Fig. 7 is an example of a General Arrangement Drawing that was created with the White Space Algorithm disabled.

Fig. 8 is an example of a General Arrangement Drawing that was created with the automatic White Space Algorithm enabled.

Fig. 9 shows the bitmap representation of a committed drawing.

DETAILED DESCRIPTION OF THE INVENTION

[0009] The invention provides support for the creation of computer-based drawings that are free from overwritten graphics and text. At the most fundamental level, this is achieved through the representation of graphics as a pair of bitmap images. As graphic elements are placed onto a drawing plane they contribute to the composition of a drawing's 'committed' bitmap and this is stored in an array of long words (32-bits or pixels per long word). The proposed placement of additional graphics is represented by a second local 'tentative' bitmap. The tentative bitmap is created only within the range of applicable long words needed. The white space algorithm uses efficient logical AND and logical OR bit manipulations to track and compare these bitmap images and determine if white space is available for placement of the proposed additional graphics. When acceptable white space is located the proposed graphics are simultaneously added to the drawing plane and inserted into the drawing's 'committed' bitmap. The pixels on the 'committed' bitmap are turned on and this becomes more densely populated as additional graphics are placed. This process is repeated until all graphics elements have been added and the drawing is complete.

[0010] The bitmaps are integer representations of real world 3D coordinates that are transformed into 2D drawing space and mapped into pixel sets. Each bit or pixel state, either a value of 1 for ON or a value of 0 for OFF, is created by stroking vectors and mapping them into an equivalent pixel space by applying the standard Bresenham line drawing algorithm. The resulting bitmaps are stored as arrays of 32-bit words and for a drawing plane of 1024mm x 1024mm a resolution of 1mm is achieved with a modest 128kilobytes (32k Long Words) of memory. This resolution is considered optimal for the class of drawings created by the inventors and can be increased or decreased as required. Increasing the resolution to 0.5 mm requires 4 times the memory while decreasing the resolution to 2mm requires only ¼ of the memory. In general, the accuracy can easily be enhanced by a factor of n with a corresponding increase in memory of n^2. Additionally, if the client application is creating smaller drawings such as

US $8^1/_2$ x 11, or European A4, the memory required can be reduced in proportion to the drawing area.

[0011] A similar approach is adopted for text elements but the vector stroking is extended to apply a series of parallel vectors that run from the base of the text to the top of the text with each vector being equal to the length of the text string. A dimension line is addressed as a composite element and is the product of lines and text elements. When dimension lines are committed to the drawing plane the entire composite element is mapped into pixel space. However, due to the nature of dimensioning, the arrowheads for the proposed graphics are omitted from the tentative bitmap. This is done to avoid adjacent tentative dimensions lines reporting unwarranted graphical clashes with committed dimensions. Another feature particularly useful for graphics such as dimension leader lines and setting out lines is the ability to automatically break lines as they pass over dense areas in the drawing. This has the same affect as the traditional pen up action and it effectively inhibits drawing over an area of committed graphics elements. This feature is enabled be setting the LineSkipOver control variable to TRUE and the type of result achieved is shown in Fig. 3.

[0012] The solution is bit or pixel-based and therefore an integer approximation of the actual drawing space. This is easily handled by applying tolerances and is achieved by specifying a default halo around elements. This has the effect of increasing the range of a proposed element thus ensuring that false white space is avoided. An extension of this halo option enables graphics proximity to be controlled by a client application. Four halo variables are provided at the Top, Bottom, Left and Right. These effectively increase the range of a proposed graphics element, defined by BlockB and BlockD, for which white space is being sought on the drawing plane. These variables are shown for a text element in Fig 2.

[0013] In dense drawing areas it may be inevitable that some degree of overwriting is unavoidable, not only for an automatic computer-based process but also to the trained eye of a qualified human draughtsman. Because of this, the algorithm supports the notion of

fuzzy logic to allow for less than perfect best-case solutions to be accepted. The algorithm provides status variables that define the quality of white spaced found and these are used by client applications to adjust preferred coordinates before committing to a drawing plane. In this manner overwriting is minimized and the need for subsequent manual clean up of automated graphics can be reduced or even avoided.

[0014] The use of sets of bitmaps to represent different aspects of graphics placements allows for flexible placement options. This includes the notion of a 'must hit' bitmap that can be defined as a filled polygon area that specifies a region of the drawing for placement of proposed graphics. The 'must hit' bitmap is used to ensure maximum overlap for the proposed graphics. This is achieved by inverting the objective when applying the bit level comparisons to the applicable pair of bitmaps. An example of where this could be used is the annotation of cartographic maps where text is required to represent a State Park and where this must be located within the park perimeter.

[0015] The following tables show the variables used by the White Space algorithm and client applications. Table 1 shows the control variables, Table 2 shows the status variables and Table 3 shows the adjustment scheme options.

Control Variable	Type	Description
TrackWhiteSpace	Boolean	When set to TRUE by a client application this enables White Space Tracking
CheckWhiteSpace	Boolean	When set to TRUE by a client application this allows placement of tentative graphics and invokes checking for White Space
LineSkipOver	Boolean	When set to TRUE by a client application the white space algorithm places a line that skips over committed graphics without drawing. This is equivalent to a traditional pen up action.
SkipGap	Double	This defines the gap to be applied for lines are placed with skip over enabled.
HaloTop	Double	This is set by a client application to define the active halo above text elements. The default value is 0.5 mm.
HaloBtm	Double	This is set by a client application to define the active halo below text elements. The default value is 0.5 mm.
HaloLeft	Double	This is set by a client application to define the active halo to the left of text elements. The default value is 0.5 mm.
HaloRight	Double	This is set by a client application to define the active halo to the right of text elements. The default value is 0.5 mm.
Scheme	Long	This is used to define the adjustment scheme to be used. The meaning of each value is shown in Table 3.
SlideInc	Double	This is used to define the incremental distance to be used when applying adjustments along the active slide vector.
SlideMaxD	Double	This is used to define the maximum offset from the initial locations that is allowed when applying adjustments along the active slide vector.
SlideType	Long	This is used to specify the use of an active slide vector or a curved path as follows: 0=None; 1=svX; 2=svY; 3=svS; 4=curved path
svS(0 to 2)	Double	This is used to specify a supplemental slide vector ie. other than svX and svY.

Table 1 – Control Variables

Control Variable	Type	Description
RotationStart	Long	This is used to specify the start angle in degrees when making adjustments by rotation.
RotationEnd	Long	This is used to specify the end angle in degrees when making adjustments by rotation.
RotationInc	Long	This is used to specify the incremental angle in degrees when making adjustments by rotation.
Radius	Double	This is used to specify the radius from the origin point to the placement point when making adjustments by rotation.
TextSplit	Boolean	This is used to control when long lines of text may be split into a multi-line equivalent. TextSplit is set to TRUE to enable splitting into multi-lines.
Npts	Long	This is used to specify the number of points defined in the Ls array and is used to make adjustments along a curve type element.
Ls()	Double	This is used to define an array of coordinates for a line string and is used to guide adjustments along a curved path.
CurveType	Long	This is used to define the curve type that is represented by the Ls array. The following values are applicable: 1 = straight line segments to be used 2 = Simpson's rule to be applied

Table 1 – Control Variables (Continued)

Status Variable	Type	Description
BlockReds	Long	This defines the total number of pixels that clash between the proposed bitmap block and the bitmap of the committed drawing graphics.
BlockB	Long	This defines the Breadth in pixels of the representative bitmap for the proposed block currently being checked.
BlockD	Long	This defines the Depth in pixels of the representative bitmap for the proposed block currently being checked.
BlockArea	Long	This defines the Area in pixels of the representative bitmap for the proposed block currently being checked.
WorstLineReds	Long	Several parallel lines are stroked to represent the extent of a text element and this value is the maximum number of pixel clashes on the worst hit. For vector elements, this is equivalent to BlockReds.
FullLineRed	Boolean	This is set to TRUE when a complete line of pixels in the proposed graphics element clashes with committed drawing graphics.

Table 2 – Status Variables

Scheme	Type	Description
1	Long	Used to apply uni-directional adjustments to a preferred location point. Adjustments are defined by SlideInc up to a maximum offset defined by SlideMaxD along either a slide vector (svX, svY or svS) or along a curved path specified by the coordinates in the Ls array. The value of SlideInc can be +tive or –tive and the maximum offset distance is based on absolute values.
2	Long	Used to apply bi-directional alternated adjustments either side of a preferred point. Adjustments are defined by SlideInc up to a maximum offset defined by SlideMaxD along either a slide vector (svX, svY or svS) or along a curved path specified by the coordinates in the Ls array. Only absolute values of SlideInc and SlideMaxD are used with this scheme.
3	Long	Used to apply a radial adjustment around a rotation point at a distance defined by Radius. The start angle is defined by RotationStart, the incremental rotation angle is defined by RotationInc and the end angle is defined by RotationEnd. The valid range is 0 through 360 degrees.
4	Long	Used to apply dimension line adjustments relative to the default secondary slide vector (svY). The adjustments are applied alternately to throw the dimension line onto opposite sides of the target dimension point. The initial specified minimum offset is incremented by a value defined by SlideInc up to a maximum defined by SlideMaxD.

Table 3 – Adjustment Scheme Options

[0016] A function is also provided to reset or initialize the 'committed' bitmap and this is invoked when starting a new drawing. This does not have any parameters and is invoked as follows:

ResetWhiteSpace()

[0017] The White Space Algorithm maintains status variables that support adjustments to an initial preferred placement coordinate. Various built-in schemes have been devised based on general requirements and are based on slide vectors, curved paths and rotation angles. A slide vector can be defined in any direction but in general two specific cases have proved effective. The primary slide vector (svX) is defined along the axis of a proposed vector/line or by the character angle of proposed text element. A secondary slide vector (svY) is defined at a 90 degree angle to the primary slide vector.

Thus if	svX = (c,s)
then	svY = (-s,c)
Where	s = sin(Angle) and c = cos(Angle)

[0018] Adjustment schemes are applied to monitor White Space status variables and apply appropriate coordinate adjustments along the X and Y slide vectors or along an application defined supplemental vector svS. Within the constraints of an application defined maximum offset, adjustments along a slide vector are repeated until adequate white space is located. A similar scheme has been utilized to affect an adjustment based on a rotation angle about a given fixed point as depicted in Fig. 6. Furthermore, client applications can directly access the status variables reported by the White Space Algorithm to apply custom adjustments. The White Space solution paradigm is flexible and should accommodate any conceivable adjustment scheme needed by a client application.

[0019] The following pseudo code shows how a client application may utilize the white space algorithm:

```
Ws.TrackWhiteSpace = TRUE
Ws.Scheme = 2
Ws.SlideType = 1
Ws.SlideInc = CharHt
Ws.SlideMaxD = 10 * CharHt
Ws.CheckWhiteSpace = TRUE
PlotString( x, y, CardinalPoint, TextString )
```

[0020] With this code example, the preferred placement coordinate (x,y) will be adjusted as required along the implied slide vector svX to ensure the text string is placed in white space. The concept of cardinal point is introduced to provide different placement options for text. These are defined relative to a notional box that encompasses the text elements and can have a value of 1 through 9 as shown in Fig. 5. The (x,y) coordinate in the PlotString function defines the location of the active cardinal point in the drawing plane.